EXPRESS MAIL LABEL NO.: EL692235275US
PATENT APPLICATION
DOCKET NO.: SRT-009
(5049/15)

A METHOD AND APPARATUS FOR EFFICIENTLY MOVING PORTIONS OF A MEMORY BLOCK

Field of the Invention

The present invention relates generally to computer memory and more specifically to transferring portions of a computer memory block.

Background of the Invention

Transmitting data from one component of a computer system to a second component of a computer system is typically an important aspect in the execution of tasks. If the data needed by the first component of a system resides on the second component and is of a substantial size, the system resources, such as the Central Processing Unit (CPU), are "tied up" (i.e., unavailable) for the period of time required to move the data. The unavailable time is often burdensome to the execution of tasks and can sometimes be critical to the performance of the computer system.

To transfer data efficiently, computer systems typically employ a Direct Memory Access (DMA) controller to transfer data from a source location to a target location without the intervention of the CPU. Further, a computer system may have multiple DMA controllers that each operate independently to transfer data between multiple I/O devices and memory. The multiple DMA controllers generally transfer blocks of data having a specific size and at a particular transfer rate.

However, the multiple DMA controllers can still generally be inefficient when transferring blocks of data at a particular transfer rate. First, transferring data of a particular size between one I/O device and memory at a particular transfer rate can be inefficient in that the data

can have a substantial size and consequently slow the operations of the computer system. Additionally, one DMA controller transfers data having the specific size between a particular block of memory and an I/O device. Moreover, when the data block has a size that is large enough to require multiple DMA transactions, the time to complete the transaction is increased.

Summary of the Invention

The present invention relates to a method and system for transferring portions of a memory block. In one aspect, the method includes the steps of configuring a first data mover (DM) with a first start address corresponding to a first portion of a source memory block and configuring a second DM with a second start address corresponding to a second portion of the source memory block sized differently from the first portion. The method also includes the steps of transferring the first portion of the source memory block by the first DM and transferring the second portion of the source memory block by the second DM. In one embodiment, the method also includes configuring the first DM with a first end address corresponding to the first portion of the source memory block and configuring the second DM with a second end address corresponding to the second portion of the source memory block.

In another aspect, the system includes a first DM and a second DM in communication with the first DM over a DM communications bus. The system also includes a first memory component having a first portion and a second portion sized differently from the first portion. The first memory component is in communication with the first DM and the second DM over a first DM-memory bus. The system additionally includes a second memory component in communication with the first DM and the second DM over a second DM-memory bus. The first DM transfers the first memory portion to the second memory component over the first DM-memory bus at a first data transfer rate and the second DM transfers the second memory portion

to the second memory component over the second DM-memory bus at a second data transfer rate.

Brief Description of the Drawings

The advantages of the invention described above, together with further advantages, may be better understood by referring to the following description taken in conjunction with the accompanying drawings. In the drawings, like reference characters generally refer to the same parts throughout the different views. Also, the drawings are not necessarily to scale, emphasis instead generally being placed upon illustrating the principles of the invention.

Fig. 1 is a block diagram of an embodiment of a computer system constructed in accordance with the invention;

FIG. 2 is a flow chart depicting the operation of an embodiment of the present invention; and

Fig. 3 is a block diagram of an exemplary embodiment of a memory component in accordance with the invention.

Fig. 4 is a block diagram of an embodiment of a fault-tolerant computer system constructed in accordance with the invention;

Fig. 5 is a flow chart illustrating an embodiment of the steps performed in the brownout phase and blackout phase by a fault-tolerant computer system in accordance with the invention; and

Fig. 6 is a flow chart illustrating embodiments of the steps performed by a fault-tolerant computer system in data move operation in accordance with the invention.

Detailed Description of the Invention

Fig. 1 depicts an embodiment of a computer system 4 that includes a first data mover (DM) 8(a) and a second DM 8(b) (generally 8). The first DM 8(a) communicates with a first memory block 12(a), or source memory block, and a second memory block 12(b), or target memory block, (generally 12) over a first DM-memory bus 16(a). The second DM 8(b) communicates with the memory blocks 12, or memory components, over a second DM-memory bus 16(b). Additionally, the first DM 8(a) communicates with the second DM 8(b) over a DM communications bus 20.

The DMs 8 substantially simultaneously transfer memory portions (not shown), or memory pages, having different sizes from the source memory block 12(a) to the target memory block 12(b). In one embodiment, the DMs 8 transfer the memory portions at different data transfer rates. In one embodiment, each DM 8 is a DMA engine, or DMA controller. As an example, a DMA controller could be used to copy data when an I/O device, such as a data logger, needs to save a large amount of data when some event occurs (e.g., the temperature of the system exceeds a predefined temperature). In a further embodiment, the DM 8 is a burst mode DMA, which transfers an entire block of memory to a specific destination. The burst mode DMA obtains exclusive access to the DM-memory bus 16 for the duration of the transfer. In yet another embodiment, the DM 8 is a flyby DMA, which executes a read and write cycle simultaneously. The flyby DMA reads data from the source address and writes the data to a target address concurrently. For example, a flyby DMA copies data from the source memory block 12(a) to a first-in first-out (FIFO) port. More specifically, the source address (i.e., a pointer to an address within the source memory block 16(a)) increments on each transfer, while the target address always refers to the same FIFO.

In one embodiment, the DMs 8 are located on an Application Specific Integrated Circuit (ASIC). Other examples of the location of the DMs 8 include, without limitation, a field-programmable gate array (FPGA), a programmable array logic (PAL), a programmable logic device (PLD), an Input/Output (I/O) board, a digital logic circuit, and the like.

To enable the transfer of memory portions at different data transfer rates, the first DM 8(a) may operate at a faster clock speed than the second DM 8(b). In another embodiment, to enable the transfer of memory portions at different data transfer rates, each DM-memory bus 16 transfers data at a different bandwidth relative to the bandwidth of the other DM-memory buses 16. For example, one of the DM-memory buses 16 may operate at 33 MHz while another DM-memory bus 16 operates at 66 MHz.

In one particular embodiment, one or all of the DM-memory buses 16 are a Peripheral Component Interconnect (PCI) bus, which is a local bus used for 8 bit or 64 bit computer system interfacing and was developed by Intel Corporation of Austin, Texas. Other examples of the DM-memory buses 16 include, without limitation, an Industry Standard Architecture (ISA) bus, an Extended ISA (EISA) bus, a Nu Bus developed by Apple of Cupertino, California, a MicroChannel Architecture (MCA) Bus developed by IBM Corporation of Armonk, New York, a Video Electronics Standards Association (VESA) bus, a VESA local (VL) bus, and the like. Additionally, the DM communications bus 20 is an internal bus to the computer system 4. In one embodiment, the DM communications bus 20 is an Inter-IC (I2C) bus, manufactured by Philips Semiconductors of New York, New York.

The source memory block 12(a) can be volatile memory components or non-volatile memory components. The target memory block 12(b) is typically volatile memory components. Examples of the volatile memory blocks 12 include, without limitation, Random Access Memory

(RAM), Static RAM (SRAM), and Dynamic RAM (DRAM). Examples of non-volatile memory blocks 12 include, without limitation, Read Only Memory (ROM), Programmable ROM (PROM), Erasable Programmable ROM (EPROM), and the like. The first memory portion and the second memory portion of each memory block 12 has a start address and an end address and can be mapped to contiguous or non-contiguous addresses.

In another embodiment, the computer system 4 includes a third DM 8(c) (shown in phantom) in communication with a third memory block 12(c) (shown in phantom). The third DM 8(c) communicates with the third memory block 12(c) over a third DM-memory bus 16(c). Although the computer system 4 depicted in Fig. 1 has three DMs 8 and three memory blocks 12, any number of DMs 8 and/or memory blocks 12 may be included in the computer system 4.

A flow chart depicting the operation of an embodiment of the DM 8 is shown in Fig. 2. The first DM 8(a) is configured (step 204) with a first start address corresponding to the first memory portion of the source memory block 12(a) to be moved. In one embodiment, the first DM 8 is configured to be a master DM 8. The master DM 8(a) is a DM 8 that initiates a data move and communicates to the other DMs 8 (e.g., second DM 8(b)), also called slave DMs, to start the data move. The master DM 8(a) also communicates additional information to the slave DMs 8(b), such as the addresses of the memory portions that each slave DM 8(b) moves. In a further embodiment, the master DM 8(a) is configured with an end address for the data move.

The second DM 8(b) is configured (step 208) with a second start address corresponding to the second memory portion of the source memory block 12(a) to be moved. More specifically, the master DM 8(a) communicates the second start address to the slave DM 8(b) over the DM communications bus 20. The communication of a start address to the slave DM 8(b) starts the data move operation. In a further embodiment, the master DM 8(a) communicates

a next address (as the second start address) and an offset address corresponding to the second memory portion. The slave DM 8 combines the next address with the offset address to obtain a particular start address for the respective move by that DM 8. In one embodiment, the second memory portion is sized differently from the first memory portion.

In another embodiment, the master DM 8(a) communicates (step 212) a "go" command to the DMs 8 to start the data move operation. In further embodiments, the master DM 8 starts a counter when transmitting the "go" command to enable synchronization of the DMs 8. The counter is reset each time all of the DMs 8 complete the move of their assigned memory portion of the source memory block 12(a).

The master DM 8(a) then transfers (step 216) the first memory portion of the source memory block 12(a) to the first memory portion of the target memory block 12(b) and the slave DM 8(b) transfers (step 220) the second memory portion of the source memory block 12(a) to the second memory portion of the target memory block 12(b).

The master DM 8(a) and the slave DM 8(b) then synchronize (step 224) their activity before performing the next data move. In one embodiment, each slave DM 8(b) communicates a message over the DM communications bus 20 to the master DM 8(a) after completing their respective data move operation. For example, the DMs 8(b) output a low value on a particular line on the DM communications bus 20 after completing their respective data move operation. In another embodiment, each DM 8 checks the counter that the master DM 8(a) started upon communication of the "go" command to ensure that all DMs 8 have completed the data move of their respective memory portions. By checking the counter, the DMs 8 synchronize so that the DMs 8 transfer data substantially simultaneously (i.e., no DM 8 starts to move the next memory portion until the other DMs 8 complete the current data move). Additionally, the DM 8 reads a

Blocked Boundary Window (BBW) to ensure that an I/O device is not substantially simultaneously transferring data to the same memory portion that a DM 8 is reading from during a data move operation.

Next, the master DM 8(a) determines (step 228) if the data move for the entire source memory block 12(a) is complete. For example and in one embodiment, the master DM 8(a) determines that the data move is complete when the next address is greater than an end address. If the data move operation is not complete, the master DM 8(a) and the slave DM 8(b) are configured in steps 204 and 208 with different start addresses that corresponds to different memory portions of the source memory block 12(a) (i.e., to move memory portions that were not previously moved). The data move operation repeats itself with respect to these different memory portions until the master DM 8(a) determines that the data move is complete for the source memory block 12(a). Following this determination, the master DM 8(a) transmits (step 232) a "stop" command to the slave DMs 8 over the DM communications bus 20.

Referring to Fig. 3, each DM 8 moves the data in the source memory block 12(a) to another location (e.g., the target memory block 12(b)). In this exemplary embodiment, the source memory block 12(a) is subdivided into four distinct memory portions 44(a), 48(a), 49(a), 50(a), each having a different size (i.e., 3 kilobytes (KB), 5 KB, 2 KB, and 6 KB). In a further embodiment, the target memory block 12(b) has corresponding memory portions 44(b), 48(b), 49(b), and 50(b) (not shown).

Each memory portion 44(a), 48(a), 49(a), 50(a) is defined by the offset and a chunk end address. In one embodiment, the start address 304, which is shown to be the start of the source memory block 12(a), is substantially equivalent to the first offset 308. The first chunk end address 312 is the end address of the first memory portion 44(a). Although Fig. 3 shows the

memory portions 44(a), 48(a), 49(a), and 50(a) having specific sizes, any sized memory portions 44(a), 48(a), 49(a), and 50(a) (i.e., different size or substantially equivalent size) can be moved by the DMs 8. As an example of a data move operation, the master DM 8(a) moves the first memory portion 44(a) of the source memory block 12(a) to the target memory block 12(b) and the slave DM 8(b) simultaneously moves the second memory portion 48(a) of the source memory block 12(a) to the target memory block 12(b) at a different data transfer rate. When the master DM 8(a) and the slave DM 8(b) complete this data move, the master DM 8(a) communicates with the slave DM 8(b) a second next address corresponding to the start of the next memory portion 44(a), 48(a), 49(a), 50(a) (e.g., the fourth memory portion 50(a)) to transfer by the slave DM 8(b). The DMs 8 continue to perform data moves until the next address is substantially greater than the end address 350.

Additionally, in other embodiments the DMs 8 move multiple memory portions 44(a), 48(a), 49(a), 50(a) in sequence in one data move. More specifically and for example, the start address 304 corresponds to the start of the source memory block 12(a) and the first chunk end address 312 corresponds to the end address of the third memory portion 49(a). The slave DM 8(b) moves the three memory portions 44(a), 48(a), 49(a) before the master DM 8(a) communicates a second next address to the slave DM 8(b). Therefore, each DM 8(a) can move any number of memory portions in any data move operation.

Fig. 4 depicts a fault-tolerant computer (FTC) system 400 in which the present invention may be used. The FTC system 400 includes a first CPU 408(a), or on-line CPU, and a second CPU 408(b), or off-line CPU (generally 408). Examples of the CPU 408 are, without limitation, a Pentium Classic/MMX CPU, developed by Intel Corporation of Austin, Texas, an AMD-K6 CPU, developed by AMD of Sunnyvale, California, and the like. The first CPU 408(a) includes

the source memory block 12(a) and the second CPU 408(b) includes the target memory block 12(b). Each memory block 12 further includes the first memory portion 44(a), 44(b) (generally 44) and the second memory portion 48(a), 48(b) (generally 48). The CPUs 408 also include an interrupt manager 412, a basic I/O system (BIOS) 416, and a CPU bus controller 420. Additionally, the CPUs 408 include a first north ASIC 424(a) and a second north ASIC 424(b) (generally 424).

The FTC system 400 shown in Fig. 4 further includes a first I/O board 440(a), a second I/O board 440(b), a third I/O board 440(c), and a fourth I/O board 440(d) (generally 440), although a FTC system can generally include any number of I/O boards. The I/O boards 440 additionally include a respective south ASIC 450(a), 450(b), 450(c), 450(d) (generally 450). Each south ASIC 450 includes a DM 8. The FTC system also includes peripheral devices 460, such as a display screen, keyboard, printers, and disk drive.

The CPU bus controller 420 communicates with the CPUs 408, the first memory block 12(a), and the second memory block 12(b). Additionally, the CPU bus controller 420 communicates with a first north ASIC 424(a) and a second north ASIC 424(b) (generally 424). More specifically, the CPU bus controller 420 communicates with the first north ASIC 424(a) over a PCI bus 428 operating at 33 MHz and communicates with the second north ASIC 424(b) over an Accelerated Graphics Port (AGP) bus 432 (i.e., an enhanced PCI bus) operating at 66 MHz. In one embodiment, the CPU bus controller 420 is the 440GX, developed by Intel Corporation of Austin, Texas.

The interrupt manager 412 manages interrupts for the multiple CPUs 408(a), 408(b). More specifically, the interrupt manager 412 transmits a hardware-generated interrupt to the CPU 408 that is most able to service that specific interrupt. In one embodiment, the interrupt

manager 412 is the I/O Advanced Programmable Interrupt Controller (APIC), developed by Intel Corporation of Austin, Texas.

The BIOS 416 is software that boots the CPUs 408 and determines what the CPUs 408 can execute without accessing a peripheral device 460. Further, the FTC system 400 (e.g., the CPUs 408) typically executes software that may be stored in non-volatile memory (i.e., ROM), which is described in greater detail below. The FTC system 400 can also have an OS. Examples of the OS include, but are not limited to, Windows NT developed by Microsoft Corporation of Redmond, WA, OS/2 developed by IBM Corporation of Armonk, New York, Netware developed by Novell, Incorporated of San Jose, California, and the like.

The first north ASIC 424(a) communicates with a first I/O board 440(a) and a second I/O board 440(b) over a first and second PCI north-south bus 444(a), 444(b) (generally 444(ab)), respectively. The second north ASIC 424(b) communicates with a third and fourth I/O board 440(c), 440(d), respectively, over a third and fourth PCI north-south bus 444(c), 444(d) (generally 444(cd)), respectively.

The first DM 8(a) and the second DM 8(b) connect to the PCI bus 428 with the PCI north-south bus 444(ab) while the third DM 8(c) and the fourth DM 8(d) connect to the AGP bus 432 with the PCI north-south bus 444(cd). The connection to buses 428, 432 that operate at different frequencies (i.e., 33 MHz and 66 MHz) enables the first and the second DMs 8(a), 8(b), respectively, to perform data moves at different data transfer rates. The peripheral devices 460 also communicate with the I/O boards 440.

As a further example, the FTC system 400 uses the DMs 8 to copy differently sized memory portions 44(a), 48(a) of the source memory block 12(a) of the on-line CPU 408(a) into

the corresponding memory portions 44(b), 48(b) of the target memory block 12(b) of the off-line CPU 408(b) prior to synchronizing the two CPUs 408.

For simplicity of explanation and depiction, the following discussion assumes that the FTC system 400 includes several components (e.g., two CPUs 408, one CPU motherboard 404), although the invention may include any number of components.

In broad overview and also referring to Fig. 5, an exemplary flow diagram is shown denoting the steps that the FTC system 400 performs in a data move operation. In one embodiment, the FTC system 400 first deactivates (step 504) the source memory block 12(a) of the on-line CPU 408(a). The FTC system 400 then uses the DMs 8 to transfer (step 508) the memory portions (e.g., memory portion 44(a), 48(a)) of the source memory block 12(a) to the respective memory portions (e.g., memory portion 44(b), 48(b)) of the target memory block 12(b). In one embodiment, the address of each memory portion 44(b), 48(b) of the target memory block 12(b) of the off-line CPU 408(b) is equivalent to the address of the corresponding memory portion 44(a), 48(a) of the source memory block 12(a) of the on-line CPU 408(a).

Simultaneously, the OS can selectively reactivate and access the memory portions 44(a), 48(a) of the source memory block 12(a) of the on-line CPU 408(a). More specifically, the OS can modify the memory portions 44(a), 48(a) of the source memory block 12(a). In one embodiment, the OS tracks (e.g., stores) the modified memory portions 44(a), 48(a).

After a certain time period, the FTC system 400 compares (step 512) the number of memory portions 44(a), 48(a) that the OS modified to a predetermined threshold. In one embodiment, the FTC system 400 compares the number of memory portions 44(a), 48(a) that the OS has modified after the DMs 8 have transferred that memory portion 44(a), 48(a) but before the completion of the data move operation. If the number of modified memory portions 44(a),

48(a) is substantially greater than the predetermined threshold, the FTC system 400 repeats the previous steps (i.e., step 504 and step 508) until the modified memory portions 44(a), 48(a) are substantially less than the predetermined threshold. When the transferred memory portions 44(a), 48(a) are less than the predetermined threshold, the FTC system 400 halts (step 516) the OS and copies (step 520) the rest of the memory portions 44(a), 48(a). After the DMs 8 transfer the rest of the memory portions, the FTC system 400 restarts (step 524) the OS. As described in greater detail below, step 504 through step 512 are referred to as the brownout phase and step 516 through step 524 are referred to as the blackout phase.

More specifically and also referring to Fig. 6, a more detailed flow diagram denoting embodiments of the steps of a data move is shown. When the CPUs 408 operate properly, the FTC system 400 performs (step 604) in a normal operation phase. In one embodiment, a voting mechanism determines (step 608) if a CPU 408 fails or is not functioning properly. If the second CPU 408(b) fails (or is not operating correctly), the FTC system 400 does not recognize the off-line CPU 408(b).

When the second CPU 408(b) does not function properly, the FTC system 400 enters (step 610) into a simplex operation phase (i.e., operating with one CPU 408). In one embodiment, the software (e.g., OS) configures the DM 8 to operate (shadow step 612) in a memory dump mode. When operating in the memory dump mode, the DM 8 transfers the data in the target memory block 12(b) of the off-line CPU 408(b) to the source, or on-line, memory block 12(a) of the on-line CPU 408(a). In one embodiment, the master DM 8 (a) obtains a write address that corresponds to the destination address for the data move (i.e., the on-line memory block 12(a)). For example, the FTC system 400 configures the DM 8 to operate in the memory dump mode to transfer data from a memory block 12 of a broken CPU 408 (e.g., a CPU 408 that

is not powered up) or a CPU 408 that is not functioning properly. More specifically, the DM 8 typically operates in the memory dump mode to determine the cause of a software crash through the examination of the memory block 12 of a broken CPU 408.

The FTC system 400 then enters (step 616) an initial blackout phase. The FTC system 400 prepares to copy memory from the online CPU motherboard 404(a) to the off-line CPU motherboard 404(b). In one embodiment, the FTC system 400 suspends user level processing.

Following the initial blackout phase, the FTC system 400 enters (step 620) a brownout phase. In this phase, the FTC system 400 copies the memory portions 44(a), 48(a) of the source memory block 12(a) of the on-line CPU 408(a) to the memory portions 44(b), 48(b) of the target memory block 12(b) of the off-line CPU 408(b). More specifically, in one embodiment the software configures the DMs 8 to operate in snarf mode (shadow step 621). The DM 8 executes in the snarf mode to determine when a peripheral device 460 writes data to the source memory block 12(a) of the on-line CPU 408(a). Because the peripheral device 460 cannot typically transmit memory writes to the target memory block 12(b) of the off-line CPU 408(b), the FTC system 400 executes the memory writes to the source memory block 12(a) of the on-line CPU 408(a) and the DM 8 (operating in snarf mode) copies these unperformed memory writes (with respect to the target memory block 12(b) of the off-line CPU 408(b)) to a separate location. In one embodiment, the separate location is a FIFO.

In one embodiment, the software then configures the DM 8 to operate in a memory update mode (shadow step 622). When operating in memory update mode, the DM 8 copies the entire source memory block 12(a) from the on-line CPU 408(a) to the target memory block 12(b) of the off-line CPU 408(b). As described above, in one embodiment the address corresponding to the source memory block 12(a) of the on-line CPU 408(a) is equivalent to the address of the

target memory block 12(b) of the off-line CPU 408(b). When executing in the memory update mode (or any other mode), the DM 8 can simultaneously operate in a secondary mode. In one embodiment, the software configures the DM 8 to operate in the master_all mode to transmit sequential memory portions 44, 48. Alternatively, the software configures the DM 8 to operate in the master_list mode to transfer and/or check non-contiguous memory portions 44, 48.

In greater detail and when operating in the master_list mode, the DM 8 obtains the start address of a list of addresses corresponding to memory portions 44(a), 48(a) that the DMs 8 move, or a modified page entry (MPE). For instance, the start address 304 acts as a pointer to the MPE. In one embodiment, the master DM 8(a) reads eight 32-bit words from the start address of the MPE to obtain the addresses of eight memory portions 44, 48 that the DMs 8 will move. The master DM 8(a) then transmits the first value, or address (e.g., start address), in the MPE to the slave DMs 8(b) and the DMs 8 move their respective memory portions 44, 48. After determining that the slave DMs 8 have completed their respective data moves, the master DM 8(a) transmits the next value in the MPE as the next address. More specifically, an example of the MPE is illustrated below:

Data Mover	MPE address
Entry is ignored	0800
*Stop Data Mover	0900
MPE	0600
**Generate interrupt	1000
MPE	0400
MPE	0300
MPE	0200
Modified Page Entry	0100

The table shown illustrates an example of the MPE including a predefined end address, denoted by *. In one embodiment, the predefined end address is substantially equivalent to 0900. The master DM 8(a) stops the data move when the address in the MPE is substantially

equivalent to the predefined end value (e.g., 0900). In a further embodiment, the master DM 8(a) generates an interrupt when the address in the MPE is substantially equivalent to a predefined interrupt address (denoted by **). In one embodiment, the predefined interrupt address is substantially equivalent to 1000.

If a CPU 408 (or peripheral device 460) writes to the source memory block 12(a) while the DM 8 is copying the memory portions 44(a), 48(a) of the source memory block 12(a), the DM 8 (which is concurrently operating in snarf mode) copies the memory write commands to a posted memory write FIFO (PMWF). The DM 8 determines when these memory writes occur by reading the BBW, as described above in Fig. 2. Further, the software updates an independent data structure (i.e., the MPE) with information about which memory portion 44(a), 48(a) was modified. During the brownout phase, the on-line CPU 408(a) operates and additionally processes commands from the peripheral devices 460.

After the DM 8 copies the source memory block 12(a) and in addition to the mode that the DM 8 is currently operating in, the software configures the DM 8 to operate (shadow step 623) in master_list mode. The software configures the DM 8 to operate in master_list mode to enable the DM 8 to copy the memory portions that the on-line CPU 408(a) modified (i.e., dirty memory portions) during the copy of the entire source memory block 12(a) (i.e., configuring DM 8 to operate in master_list mode because copying non-contiguous memory portions 44, 48).

Following the copying of the memory portions 44(a), 48(a) of the source memory block 12(a), the FTC system 400 enters (step 624) a test blackout phase. In the test blackout phase, the software evaluates the source memory block 12(a) to determine (step 628) the amount of the dirty memory portions 44(a), 48(a) not copied by the DM 8. In a further embodiment, if the amount of dirty memory portions 44(a), 48(a) not copied is greater than a predetermined

threshold, the FTC system 400 enters (step 630) additional brownout phases until the amount of dirty memory portions 44(a), 48(a) not copied is substantially less than or substantially equivalent to the predetermined threshold.

In one embodiment, the software uses a heuristic program to determine if a memory portion 44(a), 48(a) has been "dirtied" during several operations (i.e., entering additional brownout phases several times). In another embodiment, the software uses "insight" into the operating system to determine memory portions 44(a), 48(a) that are "dirtied" during several operations. For example, the software determines the address of frequently used databases.

In one embodiment, the FTC system 400 then operates (shadow step 634) in freeze mode to pause all I/O traffic. More specifically, the FTC system 400 does not accept any posted memory writes from a peripheral device 60 when the DM 8 operates in freeze mode. The software then copies or flushes the remaining states of the devices in the FTC system 400 (e.g., changes the state of the CPU 408(b) to an on-line state). When the DM 8 exits the snarf mode, the DM 8 clears the PMWF. The DM 8 then executes the memory writes that were stored in the PMWF before exiting the freeze mode (i.e., restarting the I/O traffic).

The FTC system 400 then synchronizes the execution of the CPUs 408 by validating (step 636) that the second (previously off-line) CPU 408(b) is operating in lock-step with the first CPU 408(a). In one embodiment, the software configures the DM 8 to operate (shadow step 638) in memory check mode to verify that the source memory block 12(a) of the previously on-line CPU 408(a) is substantially equivalent to the target memory block 12(b) of the previously off-line CPU 408(b). In another embodiment, the DM 8 operating in the memory check mode determines if the source memory block 12(a) of the on-line CPU 408(a) is correct (i.e., verifies that the data in the source memory block 12(a) is substantially equivalent to predetermined data).

When these checks are successful (step 640), the FTC system 400 is considered to be repaired and the target memory block 12(b) is substantially equivalent to the source memory block 12(a). The FTC system 400 consequently returns (step 644) to normal operation. If the checks are unsuccessful, the FTC system 400 again configures the DM 8 to operate (shadow step 612) in memory dump mode, as described above.

Although the configuration of the modes of the DMs 8 were described above with respect to a memory block, the software can configure the mode (i.e., memory update mode, memory dump mode, snarf mode, freeze mode, memory check mode, master_list mode, and master_all mode) of the DM 8 at any time.

Having described certain embodiments of the invention, it will now become apparent to one of skill in the art that other embodiments incorporating the concepts of the invention may be used. Therefore, the invention should not be limited to certain embodiments, but rather should be limited only by the spirit and scope of the following claims.